1. Heap (español: montón ″monton″; chinese: 堆 ″tłej″; polski: kopiec)

Przykład 1 (C++). Implementacja struktury typu sterta.

[heap.cpp]

```
#include <iostream>
#include <vector>
#include <cstdlib>                          //for random numbers
#include <ctime>                            //for random numbers

using namespace std;

class Heap{
    private:
        vector<int> k;
        int nElems;     // size of the Heap

    public:
        Heap(int maxSize=100): nElems(0){
            k.resize(maxSize);
        }

        int heapSize(){
            return nElems;
        }

        int father(int i){
            return i/2; // floor(i/2) -> double
        }

        int leftChild(int i){
            return 2*i;
        }

        int rightChild(int i){
            return 2*i + 1;
        }

        void swap(int a, int b)
        {
            int temp = k[a];
            k[a] = k[b];
            k[b] = temp;
        }

        void restoreHeap(int i){
            int left = leftChild(i);
            int right = rightChild(i);
            int max;
            if(left <= nElems && k[i] < k[left])
                max = left;
            else
                max = i;

            if(right <= nElems && k[max] < k[right])
                max = right;

            if(max!=i){
                swap(i, max);
```

```
                i=max;
                restoreHeap(i);
            }
        }

        void makeHeap(){
            int n=nElems;//k.size();
            for(int i=n/2; i>=1; i--)
                restoreHeap(i);
        }

        int pop()       // take the largest
        {
            int largest = k[1];
            k[1]=k[nElems];
            nElems--;
            restoreHeap(1);
            return largest;
        }

        void insert(int x)
        {
            ++nElems;
            k[nElems]=x;
        }

        void push(int x)
        {
            ++nElems;
            int i = nElems;
            k[i]=x;
            while(i>1 && x > k[father(i)] ){
                swap(i, father(i));
                i = father(i);
            }
        }

        void display()
        {
            for(int i=1; i<=nElems; i++)
                cout << k[i] << " ";

            cout << endl;
        }

        void sort(int t=1){
            if(t==1)
                makeHeap();

            int n=nElems;
            for(int i=n; i>=2; i--){
                swap(1,i);
                nElems--;       //decrease the heap size
                restoreHeap(1);
            }
            nElems=n;
        }
};

int main(){
    time_t aTime;
```

```
    int maxSize = 20;               //heap max size
    int heapSize = 10;
    Heap h(maxSize);                //create heap
    srand( static_cast<unsigned>(time(&aTime)) );  //seed random

    for(int j=0; j<heapSize; j++)   //fill array with
    {
        double n = rand() % 100;    //random numbers
        h.insert(n);
    }
    h.display();                    //display items
    cout << h.heapSize() << endl;

    h.makeHeap();
    h.display();

    cout << "push(77):" << endl;

    h.push(77);
    h.display();

    cout << "pop():" << endl;

    h.pop();
    h.display();

    cout << "sort():" << endl;

    h.sort();
    h.display();

      system("pause");
      return 0;
}
```

Zadanie 1 (C++). Do klasy "Heap" dodaj funkcję składową "displayTree()". Głównym zadaniem tej nowej funkcji jest wyświetlenie (na konsolę) struktury drzewiastej wszystkich danych przechowywanych w strukturze heap.

Wskazówka: zobacz bardzo podobną funkcję w klasie "Tree" (zajęcia nr 9).